|  |  |  |
| --- | --- | --- |
| Luke Pepin | CSE 4400 - P4 | 5/7/2025 |

**Task 1:**

|  |  |
| --- | --- |
| **Task** | **Commands** |
| Run the container in interactive mode | docker run --name tpm -it tpmcourse:latest |
| Generate 16 random bytes in Hexadecimal | tpm2\_getrandom 16 --hex |
| Find the largest random value that can be generated | tpm2\_getrandom 32 --hex  tpm2\_getrandom 64 --hex  tpm2\_getrandom 65 --hex |

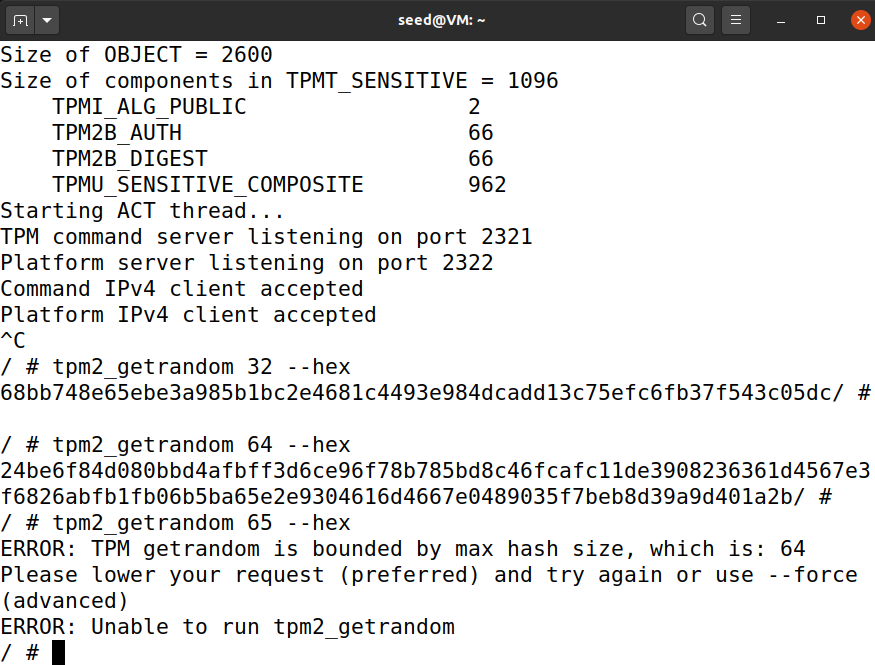
Explanation:

Random numbers can be generated with the tpm2\_getrandom command. The length of numbers generated, and the form can be determined with next arguments [length in bytes] [type], length in bytes expects an integer and type accepts empty for raw data or –hex for hexadecimal format.

This command take either 32 or 64 as its upper limit given the manufacturer of TPM. For my machine I ran 32 to see that possible upper limit then 64, given 64 returned a value that means 64 was the upper limit. Finally I ran 65 to see it fail resulting in the output ERROR: Unable to run tpm2\_getrandom.

Screenshots:





**Task 2:**

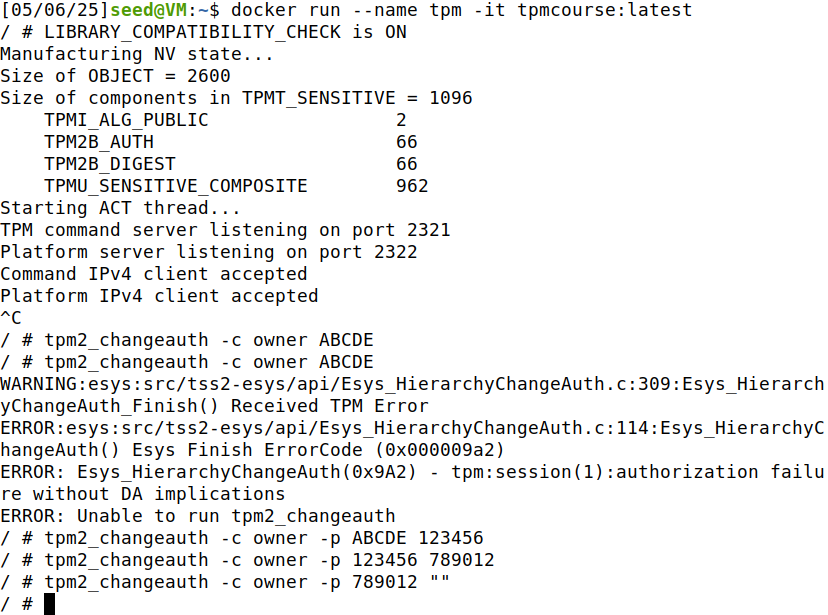
|  |  |
| --- | --- |
| **Task** | **Commands** |
| Run the container in interactive mode | docker run --name tpm -it tpmcourse:latest |
| Set the Owner’s password to “ABCDE” | tpm2\_changeauth -c owner ABCDE |
| Try the same command again | tpm2\_changeauth -c owner ABCDE |
| Change the password to “123456” | tpm2\_changeauth -c owner -p ABCDE 123456 |
| Change password again | tpm2\_changeauth -c owner -p 123456 789012 |
| Clear the password | tpm2\_changeauth -c owner -p 789012 "" |

Explanation:

Passwords for TPM hierarchies can be set and modified with tpm2\_changeauth. The command requires specifying the hierarchy being modified (owner in this case).

The experiment set a initial password with ABCDE, attempt to set the same password resulting in an error starting with WARNING (pictured in screenshot below), changing the password ABCDE-123456, changing it again 123456-78012 and finally clearing the password by changing it to “”. Note I did not both with tpm2\_clear which can be used but the changeauth solution to “” seemed more efficient.

Screenshots:



**Task 3:**

|  |  |
| --- | --- |
| **Task** | **Commands** |
| Run the container in interactive mode | docker run --name tpm -it tpmcourse:latest |
| Create Primary Key | tpm2\_createprimary -C o -g sha256 -G rsa -c o.ctx |
| Generate AES Key | tpm2\_create -C o.ctx -g sha256 -G aes -u aes\_key.pub -r aes\_key.priv |
| Load AES Key | tpm2\_load -C o.ctx -u aes\_key.pub -r aes\_key.priv -c aes\_key.ctx |
| Persist AES Key | tpm2\_evictcontrol -C o -c aes\_key.ctx 0x81010020 |
| Generate 16-Byte IV | openssl rand -hex 16 > iv.bin |
| Create Plaintext File | echo "Luke Pepin CSE 4400 P4" > plaintext.txt |
| Encrypt Plaintext File | tpm2\_encryptdecrypt -c 0x81010020 -i plaintext.txt -o encrypted.bin --iv iv.bin |
| Decrypt to Verify | tpm2\_encryptdecrypt -d -c 0x81010020 -i encrypted.bin -o decrypted.txt --iv iv.bin |
| View Decrypted File | cat decrypted.txt |
| Test Decryption Without IV | tpm2\_encryptdecrypt -d -c 0x81010020 -i encrypted.bin -o decrypted\_no\_iv.txt |
| Restart TPM | tpm2\_shutdown -c && tpm2\_startup -c |
| Reload AES Key (If Needed) | tpm2\_load -C o.ctx -u aes\_key.pub -r aes\_key.priv -c aes\_key.ctx |
| Decrypt After TPM Restart | tpm2\_encryptdecrypt -d -c aes\_key.ctx -i encrypted.bin -o decrypted\_after\_restart.txt --iv iv.bin |
| Generate RSA Key Pair | tpm2\_create -C o.ctx -G rsa -u rsa\_key.pub -r rsa\_key.priv |
| Load RSA Key | tpm2\_load -C o.ctx -u rsa\_key.pub -r rsa\_key.priv -c rsa\_key.ctx |
| Persist RSA Key | tpm2\_evictcontrol -C o -c rsa\_key.ctx 0x81010021 |
| Create Message File | echo "TPM Signing Test" > message.txt |
| Sign Message | tpm2\_sign -c 0x81010021 -g sha256 -m message.txt -o message.sig |
| Verify Signature | tpm2\_verifysignature -c 0x81010021 -g sha256 -m message.txt -s message.sig |
| Remove Persistent AES Key | tpm2\_evictcontrol -C o -c 0x81010020 |
| Remove Persistent RSA Key | tpm2\_evictcontrol -C o -c 0x81010021 |

Explanation:

While the tpm2\_encryptdecrypt functions run indefinitely and do not successfully encrypt or decrypt, this would be the series of commands used to complete the assignment.

First, the creation of a primary key in the owner hierarchy and an AES key in the TPM would be done using tpm2\_createprimary and tpm2\_create. Next, after persisting the AES key to tpm2\_evictcontrol, a random IV is generated, and together with the plaintext file, encryption is performed using tpm2\_encryptdecrypt.

With the IV and after restarting the TPM, the key’s persistence is tested by decrypting the plaintext file using tpm2\_encryptdecrypt again. Lastly, an RSA key pair is generated using tpm2\_create, signed messages are created using tpm2\_sign, and verified using tpm2\_verifysignature.

To check the persistent handles, tpm2\_getcap is used, and then the keys are removed using tpm2\_evictcontrol.

Screenshots:





**Task 4:**

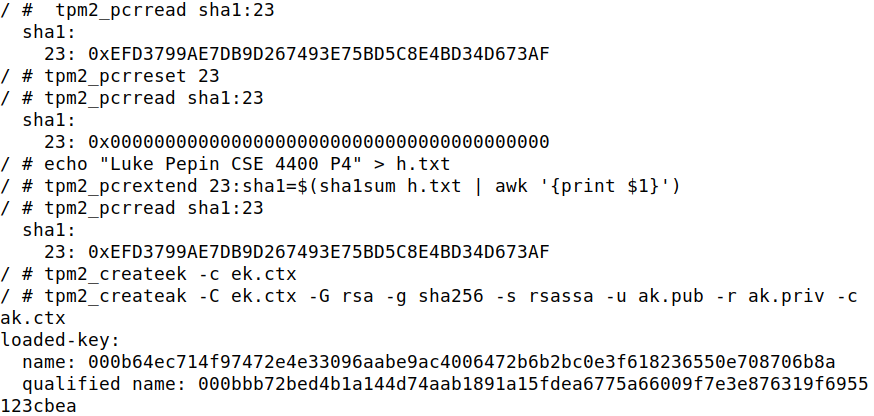
|  |  |
| --- | --- |
| **Task** | **Commands** |
| Run the container in interactive mode | docker run --name tpm -it tpmcourse:latest |
| Reset PCR 23 | tpm2\_pcrreset 23 |
| Verify PCR 23 value | tpm2\_pcrread sha1:23 |
| Create Plaintext File | echo "Luke Pepin CSE 4400 P4" > h.txt |
| Extend PCR 23 with SHA1 hash of h.txt | tpm2\_pcrextend 23:sha1=$(sha1sum h.txt | awk '{print $1}') |
| Verify updated PCR 23 value | tpm2\_pcrread sha1:23 |
| Generate Endorsement Key (EK) | tpm2\_createek -c ek.ctx |
| Generate Attestation Key (AK) | tpm2\_createak -C ek.ctx -G rsa -g sha256 -s rsassa -u ak.pub -r ak.priv -c ak.ctx |
| Generate Quote for PCR 23 (SHA1 bank) | tpm2\_quote -c ak.ctx -l sha1:23 -q nonce.bin -o quote.bin -s signature.bin |
| Restart TPM | tpm2\_shutdown -c && tpm2\_startup -c |
| Test AK Persistence After Restart | tpm2\_load -C ek.ctx -u ak.pub -r ak.priv -c ak.ctx |

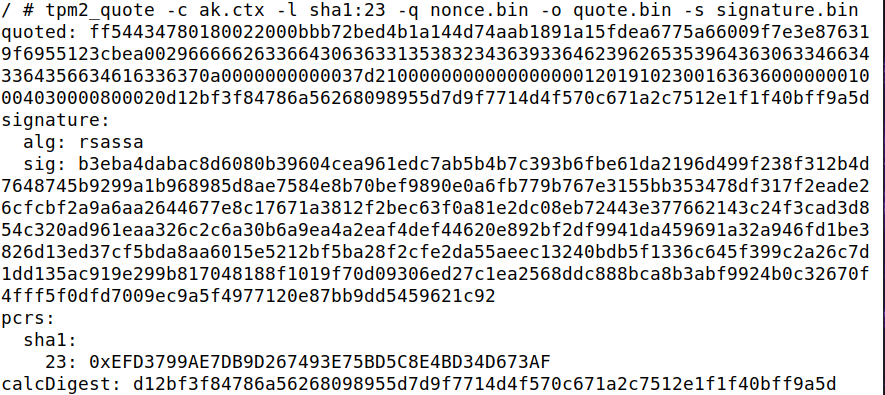
Explanation:

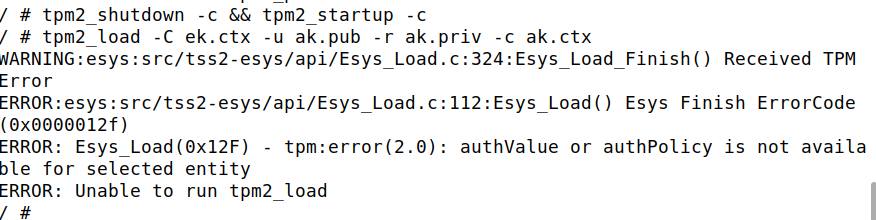
PCR 23 is first reset using tpm2\_pcrrest extnded with the SHA1 hash of h.txt using tpm2\_pcextned it is verified with tpm2\_pcread to confirm that PCR 23 is properly updated.

Next an endorsement key EK is generated using tpm2\_createek, followed by an attestation key with tpm2\_createak which is used to sign quotes. Once the AK is created the TPM quote is generated for PCR 23 in the tpm2\_quote command, after restarting the TPM the persistence of the AK is tested and failed resulting in a error since it is unusable between sessions.

Screenshots:







**Task 5:**

|  |  |
| --- | --- |
| **Task** | **Commands** |
| Run the container in interactive mode | docker run --name tpm -it tpmcourse:latest |
| Reset PCR 23 | tpm2\_pcrreset 23 |
| Create Plaintext File | echo "Luke Pepin CSE 4400 P4" > h.txt |
| Extend PCR 23 with SHA1 hash of h.txt | tpm2\_pcrextend 23:sha1=$(sha1sum h.txt | awk '{print $1}') |
| Verify updated PCR 23 value | tpm2\_pcrread sha1:23 |
| Create Policy for PCR 23 | tpm2\_createpolicy --policy-pcr -l sha1:23 -L pcr23.policy |
| Seal secret file with primary key and policy | tpm2\_create -C o.ctx -g sha256 -G keyedhash -L pcr23.policy -u sealed.pub -r sealed.priv |
| Load Sealed Key | tpm2\_load -C o.ctx -u sealed.pub -r sealed.priv -c sealed.ctx |
| Unseal the Blob | tpm2\_unseal -c sealed.ctx -o unsealed\_secret.txt |
| Modify PCR 23 and Attempt Unsealing | tpm2\_pcrextend 23:sha1=$(sha1sum different\_file.txt | awk '{print $1}') |
| Verify Unsealing Failure | tpm2\_unseal -c sealed.ctx -o failed\_unseal.txt |
| Restart TPM | tpm2\_shutdown -c && tpm2\_startup -c |
| Verify Unsealing After Restart | tpm2\_unseal -c sealed.ctx -o unsealed\_after\_restart.txt |

Explanation:

While the Blob was unable to be unsealed the reasoning and commands are sound the following occurred (or would have).

Like before PCR 23 is reset using tpm2\_pcrreset, extended with the SHA1 hash of h.txt using tpm2\_pcrextend. Next, a policy is created using tpm2\_createpolicy, binding the sealed object’s access conditions to PCR 23.

Once the policy is created, a sealed blob is generated using tpm2\_create, then loaded into the TPM with tpm2\_load. [Break from actual] The blob can then be unsealed using tpm2\_unseal, but only if PCR 23 retains the expected hash value from h.txt. It was expected that the blob unsealing would only fail if PCR 23 is modified or does not match the required, demonstrating security.

Finally, after restarting the TPM using tpm2\_shutdown and tpm2\_startup, the blob remains accessible, showing it is persistent as long as PCR 23 is.

Screenshots: (Next Page)

